OMAP3530 的启动程序——OclO

开发笔记

Rev 1.80

1 OclO 概述

OclO 是基于 TI 发布的 EVM3530 开发板的 03.00.01.06 版本的 PSP 包中的 x-loader 修改而成的。x-loader 是基于 u-boot 精简而成的，OMAP3530 内部只有 64KB 的 SRAM，不能存下整个 u-boot，因此才会先启动 x-loader，把外部的 RAM 初始化好，再把 u-boot 从 Nand、OneNand 或 SD 卡中读出来运行，这样，相当于启动过程分为 3 级：1、ROM Code；2、一级启动代码 x-loader；3、二级启动代码 u-boot。但是，原版的 x-loader 只实现了加载 u-boot 这一功能。实际上，x-loader 是一个很好的模板，很容易增加其他功能，OclO 就是这样诞生的，并且希望能保持 x-loader 的简洁程度。

1.1 关于 OclO 名字的由来

OclO 的意思是“Octagram 的 loader”，其中 Octagram 就是我的英文名。

1.2 OclO 的版本号

OclO 的版本号分为主版本号和次版本号，例如 1.80。每个主版本号中预计要实现哪些功能可以从下面的粗斜体字部分查到，而次版本号表示这些计划的完成进度，“1.80”表示版本 1 的所有预计功能完成了 80%。

目前版本号：1.80

版本 1 要完成的功能：

（1）增加一个启动菜单。

（2）OclO 运行时先倒计时，采用串口作为终端设备，在倒计时中如果按下任意键，则进入启动菜单；如果没有任何输入，则自动加载按以下顺序搜索并加载 u-boot：SD 卡 > Nand Flash。

（3）在启动菜单中增加功能[B]，启动 u-boot.bin。

（4）在启动菜单中增加功能[O]，加载用户自定义的程序，这样，OMAP3530 就可以当成单片机来玩了。

（5）在启动菜单中增加功能[D]，将 OMAP3530 配置成一个 DSP，这样就可以在 CCS 下进

行 DSP 实验了，和真的 DSP 没有什么区别。

2 OclO 使用说明

（1）将编译产生的文件 MLO 放入 SD 卡的 fat32 主分区，或存入 Nand Flash 的 x-loader 分区，并设置好 OMAP3530 的启动引脚。

（2）OclO 启动时，会倒计时，在倒计时结束前，按下 A~Z、0~9、标点等的任意键可进入启动菜单；否则将自动启动 u-boot。

（3）启动菜单中，输入 B 或 b 将启动 u-boot。

（4）启动菜单中，输入 O 或 o 将加载自定义的程序，加载地址 0x80008000。

（5）启动菜单中，输入 D 或 d 将进入 DSP 模式（此功能还未完成）。

3 OclO 编译说明

（1）在终端中将路径切换到 OclO 的目录下，例如：

/home/octagram/workspace/ARM/OMAP3530/OclO。

（2）配置：

make Ocsom3530_config

（3）使用下面命令可以开始编译，编译结束后将产生一个文件 MLO，这个是 OclO.bin 文件加上信息头后的产生的文件。

make

（4）若要删除编译产生的文件

make clean

（5）如果要将产生的文件、配置信息等全部删除（压缩备份前使用）

make distclean

4 OclO 目录说明

OclO 是由 u-boot 的修改而来的，所以大部分文件夹或文件功能是相同的，打开 OclO 的目录，可以看到以下文件夹（蓝）和文件（黑）：

- board：和具体板子有关的源代码；
- cpu：和 cpu 有关的源代码；
- disk：和磁盘分区有关的源代码；
- doc：文档；
- drivers：驱动；
- fs：文件系统；
- include：头文件；

- lib：一些库函数，包括“除以 0”错误处理、printf 函数等等；
- menu：启动菜单有关的代码；
- scripts：一些脚本；
- arm_config.mk：编译配置文件，其作用是给 gcc 增加两个编译选项；
- config.mk：最主要的编译配置文件，其作用是生成 gcc、ld 等工具的编译或链接选项；
- Makefile：工程项目管理文件；
- mkconfig：配置生成脚本，“***make Ocsom3530_config***”命令执行的操作就是该脚本；
- signGP：该程序在生成的 OclO.bin 上增加简单的文件头，生成可用于 OMAP3530 启动的文件 MLO；
- signGP.c：上一个程序的源代码，注意这个是 PC 的程序，需要 PC 的 gcc 来编译，生成的程序命名为 signGP；

5 OclO 的启动流程

（1）程序启动时的第一个文件是位于 /cpu/omap3 目录下的 start.S，该程序为 OclO 的第一阶段，为汇编语言编写；

（2）在 start.S 中会调用 /board/Octagram/Ocsom3530/platform.S 中的 G_ALowLevelInit 函数，该函数又会调用 C 语言函数 S_Init（位于/board/Octagram/Ocsom3530/Ocsom3530.c 中），该函数会依次做下列工作：

- 关闭看门狗；
- 解锁存储器，这是 OMAP 的保护机制，不解锁，将无法写一些寄存器；
- 配置各引脚的模式（大多数引脚是功能复用的，具体信息在《OMAP35x TRM》的第七章表 7-4（772 页），配置内容需查看/board/Octagram/Ocsom3530/mux.h）；
- 配置各模块需要的时钟，配置方案如下：$f_{MPU}=f_{DPLL1}$=500MHz、$f_{IVA}=f_{DPLL2}$=360MHz、$f_{CORE}=f_{DPLL3}$=332MHz、f_{DPLL4}=432MHz；
- 配置外部 RAM。

（3）之后会建立各种模式下的堆栈，然后会跳入 OclO 第二阶段，第二阶段代码是用 C 语言编写的，入口函数为 /lib/board.c 中的 OclOCStage；

（4）OclOCStage 函数会初始化串口作为终端设备；初始化(One)Nand；初始化 SD 卡；之后再等待用户输入，进入启动菜单；如果等待超时，将自动在 SD 卡、OneNand、Nand 中寻找 u-boot；如果这些操作均失败，将自动挂起，等待复位。

6 从 x-loader 到 OclO 的移植笔记（以下修改要注意大小写）

准备交叉工具链，我的工具链是自己制作的，工具前缀为 ***arm-octagram-linux-gnueabi-***，如果使用如果使用比较常用的工具链 SourceryG++，前缀是 ***arm-none-linux-gnueabi-***，下载

地址

http://www.codesourcery.com/sgpp/lite/arm/portal/subscription?@template=lite

需要下载 GNU/Linux 版的，这个版本其实也是 eabi 版本的，使用 Glibc 作为 C 语言标准库，并且可以开发 Linux 系统下的程序；而 EABI 版的使用的是红帽的 Newlib 作为 C 语言标准库，通常不跑操作系统的 CPU 将使用这个工具链，例如 STM32。

（2）阅读 Makefile 文件，修改交叉工具链的前缀：

```
CROSS_COMPILE = "交叉工具链的路径"/arm-octagram-linux-gnueabi-
```

增加配置

```
Ocsom3530_config :      unconfig

        @./mkconfig $(@:_config=) arm omap3 Ocsom3530 Octagram
```

在标签“all:”下增加

```
        cp OclO.bin ../TftpDir

        ./scripts/mkMLO.sh
```

用于将生成的 OclO.bin 文件拷贝到 TFTP 服务器的目录下，并制作 MLO 文件，同时需要在“clean”命令时清楚这些文件，因此在“clean:”以及“clobber:”标签下的内容。

（3）在/board 文件夹下增加文件夹 Octagram，在 Octagram 下又增加文件夹 Ocsom3530，将/board/am3517 中的全部内容复制到 Ocsom3530 文件夹下，OclO 将基于这个进行修改。

（4）在 Ocsom3530 文件夹下有一个文件——config.mk，该文件下有个符号 TEXT_BASE，这个符号表示 OclO 的运行地址，即 OclO 需要重定向到这个地址后，其中的与位置有关的代码才可以正常运行。

（5）Ocsom3530.c 由 am3517evm.c 修改而来；新增的 mux.h 是功能复用的引脚配置文件；interrupts.c 作用是处理中断。

（6）修改 Ocsom3530 文件夹下的 Makefile，让其能正常编译两个.c 文件。

（7）为了减少错误，先不修改 start.S 文件。

（8）config 文件在/include/configs/Ocsom3530.h，我增加的几个符号：CONFIG_AUTOBOOT_DELAY、CONFIG_SYS_UND_STACK、CONFIG_SYS_ABT_STACK、CONFIG_SYS_IRQ_STACK、CONFIG_SYS_FIQ_STACK、CONFIG_SYS_MON_STACK、CONFIG_SYS_SVC_STACK，这些符号的意义在代码中都有注释。

（9）OMAP3530 的 x-loader 是基于 OMAP3430 修改的，因此在代码中会大量出现 CONFIG_OMAP3430 或 CONFIG_OMAP34XX，将其中“34”修改为“35”。

（10）配置外部 RAM 的代码位于/board/Octagram/Ocsom3530/Ocsom3530.c 中的 config_3530sdram_ddr 函数，但是这个函数只配置了 CS0，外部 RAM 只有 128MB 可用，需要增加对 CS1 的配置后才可以使用另外的 128MB，使整个 RAM 可用，RAM 的地址是 0x80000000 ~ 0x87FFFFF 和 0xA0000000 ~ 0xA7FFFFF。

（11）将/lib/board.c 文件中的 start_armboot 改名为 OclOCStage（且要也在使用到这个函数的地方修改），并增加调用 I²C 初始化函数（TPS65930 需要使用这个总线与 CPU 通信，mmc 部分的驱动要使用到，因为 MMC1 的电源是由其管理的）、调用 SD 卡初始化函数、进入启动菜单的代码。

（12）在/drivers 文件夹中增加 I²C 的驱动程序：i2c.c 以及在/include 文件夹中增加其头文件 i2c.h。

（13）原版的 mmc 驱动程序位于/cpu/omap3 的 mmc.c 中，并带有三个头文件 mmc_host_def.h 和 mmc_protocol.h，还有一个头文件在/include/asm/arch-omap3/mmc.h，但这个驱动并不适用于我的板子，原因是我的板子的 MMC1 的电源是由 TPS65930 管理，代码中缺少对 TPS65930 的配置，参考板子公司提供的代码，将 mmc_host_def.h 移动到文件夹/include/asm/arch-omap3 中，删除 mmc_protocol.h 文件，然后还要对剩下的 3 个文件进行修改。为了简单，直接复制板子公司的代码过来。

（14）启动菜单的代码，需要增加一个文件夹 menu，并需要编写 menu.c 和该目录下的 Makefile，可以参考其他文件夹下的 Makefile，并且需要在顶层的 Makefile 中增加：

```
LIBS += menu/libmenu.a
```

以增加对该文件夹下源程序的编译。

（15）在/scripts 文件夹下增加生成脚本 MLO 文件的脚本 mkMLO.sh。

（16）开始编译，出现错误提示

```
/home/octagram/软件/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/arm-octagram-linux-gnueabi-ld:
error: Source object /home/octagram/软件
/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/../lib/gcc/arm-octagram-linux-gnueabi/4.5.1/libgcc.a(_divs
i3.o) has EABI version 5, but target OclO has EABI version 0

/home/octagram/软件/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/arm-octagram-linux-gnueabi-ld:
failed to merge target specific data of file /home/octagram/软件
/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/../lib/gcc/arm-octagram-linux-gnueabi/4.5.1/libgcc.a(_divs
i3.o)

/home/octagram/软件/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/arm-octagram-linux-gnueabi-ld:
error: Source object /home/octagram/软件
/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/../lib/gcc/arm-octagram-linux-gnueabi/4.5.1/libgcc.a(_dvm
d_lnx.o) has EABI version 5, but target OclO has EABI version 0

/home/octagram/软件/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/arm-octagram-linux-gnueabi-ld:
failed to merge target specific data of file /home/octagram/软件
/ARMTools/ocotonix/ocotonix-arm-cortex-a8/4.5.1/bin/../lib/gcc/arm-octagram-linux-gnueabi/4.5.1/libgcc.a(_dvm
d_lnx.o)
make: *** [OclO] 错误 1
```

根据提示，可知是 EABI 的版本不匹配，对于 EABI（Embedded Application Binary Interface），

我的认识还是有些不足，因此下面的理解可能不对。我的理解是这样的：EABI 是应用程序调用的一种方式，我使用的编译器是 GNU/Linux EABI，适合用来编译 Linux 内核及其应用程序，其对库函数等的调用方式会与 Linux 内核有关，与普通的 EABI 不同。但 bootloader 是不依赖操作系统的程序（bare-metal application），最佳的选择是使用普通 EABI 的工具链（bare-metal toolchain）来编译，例如，上面提到的 SourceryG++的 EABI 版本。因此，若使用 GNU/Linux EABI 的工具链来编译 OclO 时，就可能发生 EABI 冲突。参考资料 http://www.codesourcery.com/archives/arm-gnu/msg02478.html。

解决的办法是把/cpu/omap3/config.mk 中的编译选项

```
-mabi=apcs-gnu
```

删除，该选项指定了 ABI（Application Binary Interface），即应用程序接口，有该选项后 GNU/Linux EABI 的工具链在连接程序时会按 GNU/Linux EABI 的方式来调用，而 OclO 不是操作系统的应用程序，因此不能有这个选项。此时编译通过了，把产生的 MLO 文件放入 SD 的 FAT32 主分区，能正常启动 u-boot 了。

（17）修改/cpu/omap3/start.S，由于原程序中的中断部分的代码不正确（可能是 x-loader 原先设计为使用 XIP 的 flash 直接运行，不需要 ROM Code 管理中断，中断向量就是标准的位于 0x0 处，并且 x-loader 也不会使用中断，所以在之后即便有问题也没有修改，而 OclO 是需要响应中断的）所以需要重新编写 start.S 文件。

（18）修改/board/Octagram/Ocsom3530/platform.S 文件，删除一些不需要的代码。

（19）重新编译后发现程序启动时豪无现象，仔细排查错误，发现错误在start.S复制中断向量到ROM Code指定的内部SRAM中断向量处。由于需要复制56个字节，而使用的指令为

```
ldmia    r0!, {r3-r10}
stmia    r1!, {r3-r10}
```

共使用了 8 个寄存器，这样对于 56 个字节需执行两次，真正复制的字节为 64 个，超出了 8 个字节，中断向量位于内部 SRAM 的最后，在内部 SRAM 之后的地址又不可访问，超出的 8 个字节会引发错误。因此，只能刚好复制 56 个字节，修改寄存器组为{r3-r9}后正常。